DRAFT

AA

# Proposal to Provide VAX UNIX system support at Berkeley

Prof. Bob Fabry
November 8, 1979
Fabry @ Berkeley

415-642-2114

## 1. Abstract

It is proposed that a group be formed at Berkeley to provide VAX UNIX support to the ARPA-funded image understanding and VLSI contractors. This group would provide a version of the UNIX operating system containing enhancements required by the contractors in the areas of paging, access to large files, interprocess communication, networking, and performance. Additional enhancements will be developed in areas to be determined based on the interests and expertise available at Berkeley and the needs of the contractors. Some support will be available to the contractors in the area of system configuring, debugging, and interfacing.

## 2. Introduction

Modifying UNIX to be responsive to large problems such as those posed by image understanding and VLSI design is a substantial and interesting engineering effort. Such an effort will lead to a number of sub-problems suitable for investigation by students and faculty and can be expected to enrich the quality of education at Berkeley for those interested in the issues associated with applied computing.

Berkeley is uniquely suited to be the home of such an effort as the result of an early commitment to UNIX, a number of continuing UNIX-based research projects, and a history of successfully developing portable UNIX-based software. A number of these activities are mentioned in the last section of this proposal.

## 3. Staffing

Primary technical guidance to the project will be provided by faculty members. Prof. Mike Powell is extremely interested in many of the issues raised by this proposal and will play a significant role in guiding it. Prof. Richard Fateman and Prof. Larry Rowe have already had a great influence on the proposal and will continue to provide advice and technical judgements related to the project, although they are both commited to other projects at the moment.

The project will utilize staff to provide continuing technical support base for students and faculty, coordination of the various sub-projects, technical support to the contractors, and a more goal-directed orientation than can be expected of most students. One high level staff person has been tentatively recruited and another is available on a consulting basis. Two additional staff members are required.

It is expected that the project will support a number of graduate students whose research interests coincide with the goals of the project. Some excellent students have already expressed interest in such work.

## 4. Initial Plan of Attack

The critical path is getting adequate paging and interprocess communications facilities. A group of technical representatives of the various ARPA contractors should meet at an early date to evaluate the Berkeley paging version of UNIX for immediate use. We believe this system will prove adequate for initial use even if some enhancements may be required later. This group should then develop a consensus on what is required for an interprocess communications mechanism and an implementation should be begun. These areas as well as many which are less time critical are discussed in detail below.

## 5. Time Scale

Assuming that the proposal finds acceptance at each of the decision points, the following time scale is projected:

November 15: final draft of this document and documentation on the Berkeley version 7 UNIX with paging to selected ARPA people.

December 6 & 7: Meeting in Berkeley with about five representatives of the ARPA image understanding and VLSI contractors and perhaps ten or fifteen interested people at Berkeley. The purpose of the meeting would be to answer questions on the current state of Berkeley's modified UNIX system, to assign priorities to various enhancements and to develop a consensus on the technical issues related to immediately required system development. This meeting can also deal with the suitability of Berkeley as the site for this work if that is still an issue.

January 15: Formal proposal to ARPA.

April: Funding starts.

## 6. Machine Availability

Arrangements must be made for VAX computer access. In the long run, it is imperative that the project should have its own VAX. A $200,000 configuration appears to be adequate. An additional $50,000 to $100,000 will be required for site preparation: power, flooring, cooling. An initial purchase by the University may be possible if ARPA can guarantee enough income over a period of years to cover the purchase, preparation, and operations costs.

We expect that some real users could and should be accommodated on the same hardware with an agreement that part of the day would be devoted to system work and part to a semi-stable production environment using the new system.

We must also be concerned with the interrim period before a VAX could be installed. It is possible that purchase lead times coupled with an inability of ARPA to make a firm commitment for several months could lead to a delivery date during next summer. We currently have three VAX's in the department, but each is purchased with a purpose which could preclude the kind of system work which is required. It is possible that arrangements could be made with the Computer Science VAX if some of the equipment ultimately destined for a new VAX could be used temporarily to augment that system.

## 7. Essential developments

Based on the VAX proposals circulated recently by Bob Sproul it is apparent that certain changes to UNIX are essential if UNIX is to be used successfully by the image understanding and VLSI design groups.

### 7.1. Paging

Version 7 UNIX was successfully modified to incorporate paging at Berkeley over the last six months by graduate students Ozalp Babaoglu and Bill Joy. The paging system has been in reliable production use since late September and is scheduled for general distribution in December.

The current version of the paged system does not page out the kernel code and data structures nor does it allow page faults on the user page tables. The operating system currently uses 55K for program and 74K plus eight bytes per remaining physical page for data. Thus, our system which currently has 1.31 megabytes of memory allows 1.15 megabytes for users and user page tables.

There is no actual limit on a user's virtual address space size other than the 31-bit size of the address. There is a set of compile time parameters which specify an upper bound in order to size certain data structures according to the anticipated use. However, as a practical matter, the size of a user's page table will limit the useful size of a virtual address space. User page

tables require four bytes per virtual page. Thus, a 16 Megabyte virtual address space would require 128K bytes for page tables which might not be tolerable. Especially in the long run, it would seem that VLSI design and image understanding will require such large virtual address spaces that the extra mechanism and overhead of handling faults on the user page tables will be worthwhile. Such a change is tentatively viewed as desirable but not urgent.

In the current version of the system each process has a code (text) segment, a data segment and stack segment. The text segment is an image of a file which is not allowed to be modified while the process is running; an initial portion of the data segment is also initialized from this file. The data segment may be grown or shrunk by system calls; the stack is extended automatically by the system. Load formats exist for both demand initialization and preloading of the text and initialized data portions of a process. Preloading is currently the default, but large processes such as the LISP compiler use a loader option to load on demand. Text pages are always shared, while data pages are instantiated as private copies belonging to a process. The use of these three segments, each with special properties, may seem ad hoc compared to the TENEX PMAP facility, but it has the important property of maintaining exact compatibility with the earlier versions of UNIX and allows existing programs to be used without change.

The quality of the paging modifications should not be underestimated. They have been written, debugged, rewritten to clean up the organization, debugged, and used in production. One more iteration will precede the distribution next month.

Existing documents explain how to bring up the paged UNIX system and document the new data structures which have been introduced. Joy and Babaoglu are committed to producing three more documents within the next few weeks: a description of the paging system suitable for general distribution, a user document explaining the subtler aspects of working effectively in the paged environment, and a design manual.

### 7.2. Access to Large Files

We are in the process of implementing two new system calls which will allow efficient access to large files. These system calls are "vread" and "vwrite" and are virtual memory versions of the standard system read and write calls. These calls have the property that on a non-virtual system they can be replaced (at a loss of efficiency) by standard read and write calls. Pages which are virtually read will only be brought from the file upon demand (or at the systems discretion), so that virtually reading a large file is a fast operation. A virtual write writes back only frames which have been modified since the corresponding vread. The vread/vwrite mechanism does not provide for sharing of writable memory, and (as implemented) does not share page frames of data which have been read from the file.

We are in contact with John Reiser of Bell Laboratories who is (independently) pursuing a scheme whereby the process address space is totally constructed of pages mapped from files. This scheme incorporates facilities such as copy-on-write pages and is very similar to the TENEX PMAP facility. There are some problems with this approach, both in terms of system overhead, and performance, and in applying it in a tranparently distributed environment. We expect to have further discussions and perhaps to incorporate features similar to these at some point.

### 7.3. Interprocess Communication

The UNIX pipes are notoriously weak as an interprocess communications mechanism. Among the problems with the standard implementation of pipes are:

Pipes cannot be named in the file system (even though they are implemented as files.)

Pipes cannot be used to communicate between processes unless the processes share a common ancestor which sets up the pipe. (This is a consequence of the previous point.)

A process cannot wait for messages to come in through several pipes nor can a process test to see if there is a message pending on a pipe without being blocked if there is no message.

Pipes are byte oriented; any structure must be imposed at the user level.

If two or more processes push bytes into the same pipe, the bytes may become interleaved, probably causing confusion.

Pipes implement a FIFO queue; it is not possible to send an interrupt down a pipe ahead of bytes already in the pipe.

A variety of modifications to UNIX have been implemented or suggested which solve one or more of these problems. A consensus must be developed as to what the correct mechanisms are.

The message-oriented ARPANET approach and the pipe-oriented UNIX approach are in some sense at the opposite ends of a spectrum. The ARPANET approach must be maintained for compatibility with existing programs and sites. The UNIX approach must be maintained for compatibility with the existing UNIX programs. An attempt to support both approaches in one mechanism could result in a horrible botch.

The interprocess communication mechanism design interacts with the higher levels of both local network designs and ARPANET protocols. A group of processes must be able to work together over either type of network using the identical interprocess communication mechanism used locally.

Although a UNIX pipe appears to be a stream of bytes, pipes are used almost exclusively to hold ascii characters. The notion of a stream of ascii characters as the interface between processes which are being combined to solve a larger problem is extremely strong because of its utter simplicity. Programs can be tested in pieces by driving them from the keyboard and they can be debugged by dumping the characters onto a terminal. The simplicity allows even novices to invent new interconnections of processes from the command level. UNIX is strong because of this cleanliness. A substantial component of the cleanliness may be the heavy use of streams of ascii characters. Can an interprocess communication mechanism be invented which will satisfy the new needs and yet not subvert this basic interface?

We believe that the (single machine) implementation of whatever interprocess communications mechanism is chosen will be relatively straightforward. We are more concerned with getting the specifications correct. Several groups have developed proposals. These should be evaluated to get a consensus on how to proceed.

### 7.4. Networking

The UNIX ARPANET software with which we are familiar is based on that done at the University of Illinois by Dr. Greg Chesson while he was a student. We have been running such a version at Berkeley for a short time. It was written in a contorted way because of the address space limitations on the 11/70 and the limitations of the UNIX interprocess communications mechanism. The code has been converted to version 7 UNIX but not to the VAX.

One approach would be a major redesign using the larger address space of the VAX, relying on the improved interprocess communications mechanism which will be developed, and borrowing from any improved versions of the code that may have been developed in the meantime. An alternative approach using a gateway PDP 11/70, PDP 11/23, or similar computer to interface to the ARPANET and to a local network should also be examined.

Although we do have considerable local network expertise at Berkeley, notably Prof. Larry Rowe and his students, an appropriate senior staff person be found for this area.

### 7.5. Performance

The issue of adequate performance was not addressed in Sproul's draft which focused on features. UNIX uses simplistic strategies in many areas. We expect that some of these strategies will prove lacking in the context of a large paged virtual address space machine trying to support solutions to large problems. Issues of page size, secondary storage allocation strategies, processor scheduling, inode management, and so on are likely to require considerable rethinking.

A number of tools for analyzing UNIX performance have been developed and used at Berkeley. We expect that these tools will be applied on a continuing basis to pinpoint areas in which substantial improvements can be obtained.

## 8. Desirable modifications.

Turning again to Sproul's recent draft, several changes have been identified as desirable but not essential. Brief comments are included on each topic he mentioned, more to expose my biases than to decide anything. Actual projects in these areas should be based on the interests and expertise at Berkeley relative to that elsewhere.

### 8.1. Terminal multiplexing.

Version 6 UNIX has no facilities which support terminal multiplexing or multiple groups of processes accessed through a single terminal. Some work was done on such facilities at RAND (VTUNIX), but this has not yet been moved to UNIX version 7.

The multiplexor mechanism implemented in version 7 UNIX by Dr. Greg Chesson supports terminal multiplexing enough to allow, for instance, a process to be written to split the display screen into two windows running a shell or other command processor in each window. This software is functional on the VAX.

For more flexible support of terminal multiplexing, a process control mechanism more powerful than that of version 7 UNIX is required. The ability to run more than one group of processes and to arbitrate terminal access between them is necessary. The process control mechanism of the ITS system at MIT is powerful and general and meets many of these requirements. J. Kulp of IIASA Austria has made some simple modifications to the (Version 6) UNIX kernel and command interpreter which implement such a scheme. He is currently reimplementing these features in the version 7 system, and we expect to put these facilities in our system soon.

Given appropriate multiplexing capabilities and process control primitives, we hope to be able to construct powerful screen management facilities. We are in contact with a number of people at Bell Laboratories who are also interested in such issues; further discussion and developments are anticipated as our experience with such mechanisms grows.

### 8.2. File System Reliability

Our experience is that the file system is quite reliable if the standard crash recovery procedures are understood by the operations people. We have had serious problems only when the system was left in a confused state after a crash. Nevertheless, further work may be required.

Prof. Mike Powell has several ideas of how to enhance the reliability of the file system by writing out information in a more timely manner. Also, we are aware of work in the UNIX Support Group at Bell Labs which we hope to obtain and which will help to make the file system even more reliable.

### 8.3. Operational Support.

We use an automatic restart program to restart after simple crashes. Automatic crash analysis programs also exist. Making such programs generally available seems quite useful.

Another area of operational support is file system dumping. The current dump programs

dump on a logical basis and take a long time as a result. This will be a bigger problem as file systems grow to support the large data bases required by VLSI design and image understanding.

## 9. Other Relevant Work at Berkeley

The areas described below indicate relevant areas of expertise at Berkeley. These are areas in which further dialog might be initiated if the ARPA contractors are interested in taking advantage of the work.

### 9.1. A Simple Local Network

A very simple local network was developed by graduate student Eric Schmidt under my direction and has been running for about a year and a half. The network allows a user to run a non-interactive process on a remote machine with its standard input and standard output connected to files or pipes on either the local machine or the remote machine. The user need not know the topology of the net. There are currently about ten machines in the Berkeley net, all running UNIX. The net is now or soon will be running at some other locations. The strength of the network is that it requires no changes to the operating system and no special hardware. (Each computer thinks the other is a terminal.) The network would be suitable for groups running several UNIX systems which are not interconnected by a more sophisticated network. It is quite adequate for low speed file transfer, mail, access to special peripherals such as typesetters and printers, and so on. It is simple and it works.

### 9.2. A Fancier Local Network

A much more ambitious project which will presumably replace the simple network between some of the machines is being developed by Prof. Larry Rowe and his student Ken Birman in connection with the Ingres project. When this project is complete, there may be some interest in the result. There are several things which make it noteworthy:

The local net is integrated into UNIX in such a way that a program does not need to know that a remote file is remote in order to use the file. UNIX system calls for dealing with files know about both local and remote files. This approach seems to be different from that currently taken by most groups and has a strong compatibility property: existing programs can access remote files without being reprogrammed.

They are using second generation commercial hardware based on the Irvine ring scheme and may have a working system which could be copied easily before some of the other systems are available.

They are exploring multi-address packet protocols and think they have some good applications for them.

### 9.3. Portable Software Development

A large body of programs have been developed at Berkeley which are routinely run on 11/34's, 11/40's, 11/45's, 11/70's, and VAX's; under both versions six and seven of UNIX; and under systems with various local modifications. The programming techniques which allow this portability have been developed over time and should be quite useful.

### 9.4. Ingres

There is a large relational data base project headed by Prof. Mike Stonebraker. This project uses UNIX exclusively and is exploring (among other things) distributed data bases.

### 9.5. Terminal Handling Support

We have projects under way to build high quality bit map terminals and/or to buy computers such as the Three Rivers PERQ. Support of such terminals is an important area for further development as is the issue of high speed networking which they inevitably involve.

### 9.6. Lisp

A substantially compatible version of PDP-10 MACLISP has been implemented by Prof. Richard Fateman, staff member Keith Sklower, students, and Bell Labs personnel. The compatibility extends to a substantial breadth and depth and includes a compiler. In the next few months they expect a fully functional MACSYMA system on the VAX, using identical source code (50,000 lines of LISP) to that on the PDP-10 and running at KA-10 speed. Scientific software research under Department of Energy sponsorship will help provide superior numerical facilities coordinated with LISP and MACSYMA development. Efforts to bring up other MACLISP programs such as FRL are underway. In an ISI feasibility study of INTERLISP migration paths to the VAX, one route will be based on the LISP system at Berkeley which is coded almost entirely in C and LISP itself.

### 9.7. Pascal

We have a Pascal interpreter which has been extensively used for instruction now available on the VAX. We are in the process of modifying the Pascal translator to generate the intermediate format of the Portable C Compiler and hope to have a Pascal compiler available on the VAX early next year. This will allow full mixing of Pascal, C and Fortran 77 code, and will allow Pascal programs to be debugged with the source language symbolic debugger sdb.